NaSTech
JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

DYNAMIC PATH PLANNING AND MAPPING USING SENSOR FUSION WITH ROS2

Hasan Erdinç KOÇER[1]* , Cem Berkay KARACA[2]

[1]Selçuk University, Faculty of Technology, Department of Electrical and Electronics Engineering, Division of Circuits and Systems, Turkey

[2]Selçuk University, Institute of Science, Department of Defense Technologies, Turkey

ABSTRACT

In this study, the dynamic path planning capabilities of mobile robots in indoor environments were investigated. The focus was on examining how newly added obstacles to an initially static map affect the robot's navigation behavior. To achieve this, the ROS2 Nav2 package along with the DWB Planner module was utilized. The Gazebo simulation environment served as the testing ground, allowing the robot to use LIDAR data to detect obstacles in real-time and navigate safely to the target without collisions. Throughout the study, three primary scenarios were established: the first involved navigating to the target on an obstacle-free map as a reference; the second examined the impact of manually added static obstacles on the robot's path planning; and the third tested how adjusting costmap parameters enabled the robot to pass closer to obstacles. Experimental results revealed that the ROS2 Nav2 framework could dynamically adapt to environmental changes, allowing the robot to re-plan its path as needed. Moreover, when the costmap parameters were reduced, the robot managed to pass closer to obstacles in a safe and controlled manner, which contributed to a decrease in both the total path length and the target arrival time. These findings highlight the direct impact of dynamic obstacles and costmap configurations on the robot's path planning performance. Overall, the ROS2 Nav2 framework has proven to be a robust and flexible solution for safe and efficient navigation in indoor environments.

Keywords: ROS2, Nav2, DWB Planner, Costmap, LiDAR,

INTRODUCTION

The ability of mobile robots to operate safely in shared environments with humans has become a rapidly evolving field of research. In such settings, robots are expected to navigate safely on a known map and also detect and avoid unexpected obstacles that may suddenly appear.

In this thesis study, a path planning system was developed on the ROS 2 platform to enable a mobile robot, equipped solely with a LiDAR sensor, to reach a target position. The LiDAR sensor scans the environment 360 degrees to detect both static and dynamic obstacles. However, this detection alone is not sufficient. In order to determine the robot's position accurately, information from various sensors must be fused and evaluated together.

In this context, sensor fusion plays a critical role. In addition to LiDAR data, information from the IMU (accelerometer, gyroscope, magnetometer) and motor encoders is integrated to estimate the robot’s position and direction of movement more accurately. This fusion process allows the robot to make safer and more flexible decisions in response to environmental changes.

The Nav2 package, which is part of the modular structure of ROS 2, was used as the navigation framework, and the DWB (Dynamic Window Approach) algorithm was selected as the local planner. The developed method ensures that the robot navigates with environmental awareness, especially in narrow indoor spaces with human presence. In this way, the study contributes to the development of low-cost and reliable autonomous navigation solutions.

2. LITERATURE REVIEW

Macenski, S., Martín, F., White, R., & Clavero, J. G. (2020). The Marathon 2: A Navigation System. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 2718–2725). IEEE.

This study introduces Marathon 2, a ROS 2-based navigation system. The system includes features such as dynamic obstacle avoidance, costmap layers, and SLAM integration. The paper explains in detail how ROS 2 utilizes navigation stacks and how costmap layers are updated in dynamic environments. Additionally, it investigates how LiDAR data is integrated into costmap layers and used in dynamic path planning. This work serves as an important reference for researchers developing dynamic path planning and obstacle avoidance strategies for ROS 2-based mobile robots.

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-time loop closure in 2D LIDAR SLAM. In 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 1271–1278). IEEE.

This paper investigates real-time loop closure techniques in 2D LiDAR-based SLAM systems. It offers significant insights into how LiDAR data can be used for costmap-based path planning in ROS 2. The study proposes new algorithms to improve the performance of SLAM and path planning systems, especially in dynamic environments. It also details how LiDAR-based SLAM systems can be integrated with ROS 2 and how these systems contribute to the costmap layers.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1), 23–33.

This article explains the Dynamic Window Approach (DWA) for collision avoidance. DWA is an effective method that enables mobile robots to navigate safely in dynamic environments. The paper provides important information on how DWA can be used with costmap layers in ROS 2 and discusses in detail how LiDAR data can be integrated into the dynamic window approach.

Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with Rao-Blackwellized particle filters. IEEE Transactions on Robotics, 23(1), 34–46.

This study addresses the processing of LiDAR data using grid mapping and Rao-Blackwellized particle filters. The paper explains how these methods, commonly used in SLAM systems, can be integrated with ROS 2 and contribute to costmap layers. It also proposes new techniques for grid mapping and path planning in dynamic environments.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., … & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In ICRA Workshop on Open Source Software (Vol. 3, No. 3.2, p. 5).

This paper presents the fundamentals of the Robot Operating System (ROS) and ROS 2. It examines in detail features such as navigation stacks, costmap layers, and SLAM integration. The study offers key insights into how ROS 2 can be used to develop dynamic path planning and obstacle avoidance strategies for mobile robots.

Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (Vol. 3, pp. 2149–2154). IEEE.

This article introduces the Gazebo simulator, which can be integrated with ROS 2. Gazebo is a significant tool used for simulating LiDAR-based dynamic path planning and SLAM systems. The paper details how Gazebo can be used with ROS 2 and how costmap layers can be tested within the simulation environment.

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic Robotics. MIT Press.

This book provides a comprehensive overview of probabilistic robotics, including LiDAR-based perception, mapping, and path planning techniques. It presents foundational knowledge for SLAM and dynamic path planning in ROS 2-based systems. The book also discusses how costmap layers can be updated using probabilistic methods in detail.

Borenstein, J., & Koren, Y. (1991). The vector field histogram—fast obstacle avoidance for mobile robots. IEEE Transactions on Robotics and Automation, 7(3), 278–288.

This paper explains the use of the Vector Field Histogram (VFH) method for fast obstacle avoidance. VFH is an effective technique for LiDAR-based dynamic path planning. The study provides important information on how VFH can be used with costmap layers in ROS 2.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011). g2o: A general framework for graph optimization. In 2011 IEEE International Conference on Robotics and Automation (ICRA) (pp. 3607–3613). IEEE.

This work presents g2o, a general framework for solving graph-based optimization problems, particularly used in SLAM and dynamic path planning. The paper describes how g2o can be applied in ROS 2-based systems and how it contributes to costmap layer integration.

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., & Konolige, K. (2010). The Office Marathon: Robust navigation in an indoor Office environment. In 2010 IEEE International Conference on Robotics and Automation (ICRA) (pp. 300–307). IEEE.

This article introduces the Office Marathon, a ROS-based navigation system. The system includes features such as dynamic obstacle avoidance, costmap layers, and SLAM integration. The study provides valuable insights into how similar systems can be developed using ROS 2.

3. MATERIALS AND METHODS

3.1 Planning Algorithms

This section discusses various algorithms developed to address the motion planning problem. Our goal is to identify where the proposed solution fits within existing planning methods and to highlight the core principles that shape these approaches. The system presented in this study adopts a path planning approach that is responsive to dynamic obstacles, directly generates trajectories, and leverages both geometric and heuristic methods. By examining the performance criteria, we can determine the type of implementation—whether it is trajectory-based or path-based. Through classification, we assess the structure, speed, and suitability of the algorithm.

3.1.1 Criteria

Several fundamental criteria are used to classify planning algorithms. These criteria help us understand the algorithm's output type, operational principles, and mathematical foundations. The main criteria are as follows:

- **Output Type:**
  The output of the algorithm may be a path, a sequence of motions (trajectory), or a symbolic representation.
  While a path defines only the positions the robot should pass through, a trajectory also specifies the velocity, timing, and method for reaching those positions.
  In this thesis study, a traditional path is not used; instead, time-based trajectories are generated in real-time according to environmental conditions.
  - **Set-based Algorithms:**
    These algorithms typically divide the workspace into segments. However, this division only creates the map; an additional algorithm is needed to determine the appropriate movement afterward.
  - **Solve-based Algorithms:**
    These algorithms directly produce a feasible motion sequence from the starting point to the goal. The DWB local planner is an example of a solve-based algorithm, which generates the most suitable trajectory in real time to reach the goal.
- **Spatio-Temporal Characteristics:**
  This refers to how the algorithm incorporates time and space during planning.
  - **Predictive Algorithms:**
    These generate a set of motion options with a forward-looking approach and determine the most optimal one over a longer time horizon. Although such methods typically offer better performance, they are computationally more expensive.
  - **Reactive Algorithms:**
    These make fast decisions based on the current situation over a short time horizon. While computationally less demanding, their predictive ability is limited. In this thesis, reactive algorithms are used to generate trajectories.
- **Mathematical Approach:**
  This refers to the mathematical methods on which the algorithm is based when solving the planning problem.
  - **Heuristic Methods:**
    Rely on experience and rule-based decision-making.
  - **Geometric Methods:**
    Use spatial data and geometry to generate solutions.
  - **Biomimetic Methods:**
    Mimic the behavior of natural systems.
  - **Logical Approaches:**
    Use predefined rules and logical relations.

These criteria help in comparing different algorithms and determining which is more suitable under specific circumstances. The criteria applied in this thesis are selected accordingly.

NaSTech
JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

Table 3.1. Fundamental Criteria Table

Criterion	Selected Type	Description
Output	Solve-algorithm	Generates trajectory
Spatio-Temporal	Reactive	Instant environmental reaction
Mathematical Basis	Geometric + Heuristic	Uses methods such as costmap, prediction, velocity

3.1.2 Classification (Taxonomy)

In this section, planning algorithms are divided into six main categories using the criteria described above.

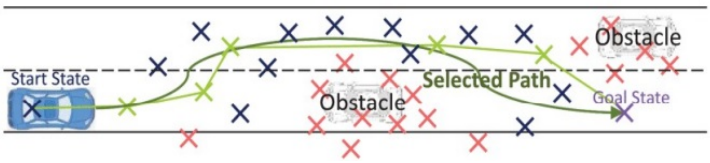
A. Space Configuration Algorithms

These algorithms segment the workspace (W) over time. They generally operate in three main steps:

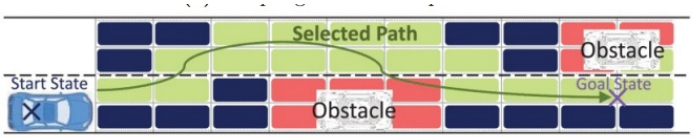
1. **Sampling / Discretization**
   The workspace is divided into points (or cells), either randomly or in a specific pattern.
2. **Elimination of Colliding / Invalid Elements**
   Points that intersect with obstacles or are kinematically infeasible are discarded.
3. **Path Finding**
   An optimal path is determined among the remaining points, forming a sequence that the robot will follow.

Within this category, the following methods are commonly used:

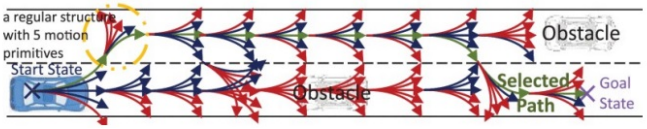
- **Sampling-Based Methods:**
  For example, the PRM (Probabilistic Roadmap Method) algorithm selects random points from the workspace and constructs a graph by connecting them while taking obstacles into account.
- **Connected Cells Method:**
  The workspace is geometrically divided into cells, and connections are made between these cells to find a path. Examples include DWA (Dynamic Window Approach) and Voronoi-based methods.
- **Lattice Representation:**
  Using motion primitives, each possible movement is linked to the next, forming a reachability graph. This method is particularly suitable for structured environments like highways.



a) Sampling-Based Decomposition



b) Connected Cells Decomposition



c) Lattice Representation

Figure 3.1. Space Configuration Types

B. Pathfinding Algorithms

These algorithms aim to find the shortest path on a previously constructed graph or structure. The most common examples include:

- **Dijkstra:**
  Finds the shortest path between two points.
- **A* :**
  A faster version of Dijkstra's algorithm that reduces computation time by exploring fewer nodes.
- **RRT and RRT* (Rapidly-exploring Random Tree):**
  These methods randomly sample points from the workspace and connect them to form a path, which is especially useful in unknown or complex environments.
  RRT* is an improved version of RRT and converges toward a near-optimal solution.

C. Attractive and Repulsive Force-Based Methods

These methods balance attractive forces that pull the robot toward the goal and repulsive forces that push it away from obstacles.

- **Artificial Potential Field (APF):**
  Applies attractive forces toward the target and repulsive forces away from obstacles. The robot follows the path of lowest energy within the potential field.
- **Elastic Band:**
  An initially generated free path is treated like an elastic band that reshapes itself to maintain a safe distance from obstacles.

D. Artificial Intelligence and Learning-Based Methods

These methods enable the algorithm to learn through interaction with the environment.

- **Logical Approaches:**
  Include decision trees, finite state machines (FSM), and Bayesian networks.
- **Heuristic Methods:**
  Use rule-based strategies derived from experience.
- **Fuzzy Logic:**
  Operates with data containing uncertainty and imprecision.
- **Artificial Neural Networks (ANNs, CNNs) and Reinforcement Learning:**
  Enable adaptation to environmental conditions by training on multidimensional data.

E. Numerical Optimization

These methods aim to minimize a cost function under a set of constraints. For example:

- **Linear Programming (LP), Quadratic Programming (QP), Model Predictive Control (MPC), and Dynamic Programming (DP):**
  These techniques seek mathematically optimal solutions within well-defined constraint sets.

F. Human-like Methods

These approaches attempt to imitate human decision-making processes. In complex situations, solutions are generated using human-like strategies (risk estimation, taxonomic models). Although a DWB-based method was preferred in this thesis, sampling-based algorithms such as RRT* offer more suitable options in unknown environments that require exploration; MPC and Lattice methods are more appropriate for scenarios such as highways that require precise maneuvers; and AI-based approaches provide better options in social or dynamic environments.

Table 3.2 Classification Table

Category	Used?	Explanation
1. Space Configuration	Yes	The DWB algorithm divides the workspace into a grid/costmap.
- Sampling-Based Decomposition	No	Sampling algorithms such as PRM, RRT, and RRT* were not used.
- Connected Cells Decomposition	Yes	DWB makes decisions based on a cell-based connectivity structure.
- Lattice Representation	No	Lattice-based motion primitives were not used.
2. Pathfinding Algorithms (Dijkstra, A*)	No	Graph-based pathfinding algorithms were not implemented in this thesis.
3. Attractive / Repulsive Forces	No	Methods such as Artificial Potential Field and Elastic Band were not used.
4. Parametric Curves	No	Curve-based methods (e.g., Spline, Bezier) were not preferred.
5. Artificial Intelligence (AI) Methods	No	AI techniques such as Fuzzy Logic, ANN, SVM, and RL were not used.
6. Numerical Optimization	Yes	The TEB algorithm fits this category as it optimizes time- and distance-based cost functions.

3.2 ROS Kavramları

ROS (Robot Operating System) is an open-source software framework developed for robots. It is not a real operating system (such as Windows or Linux), but it runs on top of them to provide some robot-specific services. These services include:

- Hardware abstraction (facilitating communication with robot sensors and motors),
- Controlling devices (e.g., operating motors),
- Communication between different software components within the robot,
- Organizing software packages.

Thanks to its open-source architecture, the ROS platform offers modular and flexible solutions for real-time control systems (Quigley et al., 2009). The core messaging structures and node management of ROS are essential concepts that must be understood for real-time applications (O'Kane, 2013).

ROS (Robot Operating System) serves as an infrastructure that functions like the brain of the robot. Although ROS1 was used for many years, with the release of ROS2, it has become a faster, more secure, and modular system.

Table 3.4: Comparison of ROS1 and ROS2

Feature	ROS1	ROS2
Communication Infrastructure	TCPROS (unidirectional, low security)	DDS (real-time, secure, bidirectional)
Real-Time Support	No	Yes
Platform Support	Limited (mostly Linux)	Wide (Linux, Windows, macOS)
Multi-Robot Support	Weak	Advanced
Lifecycle Management	No	Yes (Lifecycle Nodes)
Industrial Application Compatibility	Low	High

ROS2 is particularly preferred in systems operating in dynamic environments, such as mobile robots. Thanks to its real-time and security features, it provides more stable and reliable results, especially in indoor applications.

3.2.1 Core Building Blocks of ROS

1. **Nodes:**
   Every small program running within ROS is called a node. For example, one node may process camera images, while another controls the motors.
2. **Messages:**
   Nodes exchange data with each other by sending messages. These messages can contain simple data like numbers or text, or more complex information such as the robot's position.

3. **Topics:**
   These are communication channels through which messages flow. One node acts as a "publisher" and sends messages to a topic, while others act as "subscribers" and receive those messages. (The publish–subscribe model)
4. **Services:**
   Used when one node requests information from another. For example, a node may say "send position data", and the other responds. The process is completed, and the connection closes.
5. **Actions:**
   Similar to services but designed for long-running tasks. For instance, if it takes time for a robot to reach a target location, actions allow feedback to be received throughout the process (e.g., progress updates).

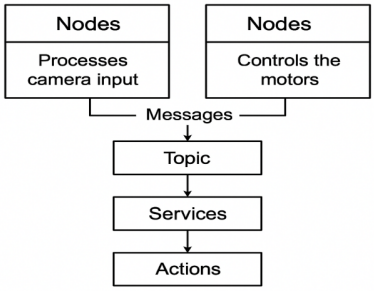


Figure 3.2. Information flow between the core components of ROS.

The ROS software is organized into **packages**. Each package contains:

- Nodes,
- Configuration files,
- Libraries, and other components.

These packages can be grouped into higher-level structures called metapackages, which combine packages with similar functions. For example, all components responsible for robot navigation can be grouped under a metapackage called Navigation Stack.

In the ROS system, a special node called the Master Node manages all communications. It coordinates how all components in the system interact with each other.

3.2.2 Navigation in ROS2 (Nav2)

IMU and odometry data obtained from the robot's own sensors serve as the primary inputs for position estimation. However, precise position information is calculated by the AMCL algorithm within ROS 2. In other words, the hardware data is processed by the software. Localization is performed using the AMCL (Monte Carlo Localization) method. Sensor data and position information are ultimately transmitted to the robot as a cmd_vel command.

The navigation packages of ROS, Nav1 and Nav2, enable mobile robots to find the safest and most efficient path to their target. While Nav1 is compatible with ROS1, Nav2 operates within the ROS2 framework and offers a much more flexible structure.

Table 3.5: Comparison of Nav1 and Nav2

Feature	Nav1 (ROS1)	Nav2 (ROS2)
Architectural Structure	Fixed structure in a single node	Modular structure (servers and behavior trees)
Planner Support	Fixed (usually DWA)	Extensible (DWB, TEB, etc.)
Robot Type Support	Limited (differential, omni)	Broad (Ackermann, differential, omni)
Map Representation	2D Costmap	2D, 3D, SLAM-compatible
Parameter Management	Static	Dynamic (lifecycle and parameter server)
Developer Support	Inactive	Active community and documentation

Nav2 enables the robot to respond more quickly in environments with dynamic obstacles. In this regard, Nav2 is much more suitable for modern applications.

The Nav2 system of ROS2 is a framework that allows the robot to move autonomously. This system operates using a Behavior Tree, which determines the order and timing of the robot's tasks. In the Nav2 system, three main tasks are managed by three separate action servers:

1. **Planner Server:**

- Generates a path from the robot's current position to the target.
- Can run different planning algorithms (e.g., A*).

2. **Controller Server:**

- Follows the path generated by the planner.
- The DWB (Dynamic Window Based) controller used in this thesis operates here.
- Adjusts the robot's speed and direction in real time and prevents collisions.

3. **Recovery Server:**

- Activates when the robot cannot reach the target.
- Performs recovery behaviors such as reversing or reorienting.

The Behavior Tree (BT) controls when and in what order these three servers operate. Each task is considered a "branch" of the tree. It ensures the sequence: plan the path, follow the path, and if an obstacle is encountered, perform recovery.

Nav2 System and Schematic Structure

```
+---------------------------+
| IMU + Odometry + AMCL    | ← Precise localization (position and orientation)
+-------------+-------------+
          ↓
+---------------------------+
|     Local Planner        |
|          DWB              | ← Path generation based on velocity, direction, and obstacle status
+-------------+-------------+
          ↓
      cmd_vel → Robot Decision
```

* Note: This structure operates under the ROS2 Nav2 system.
* AMCL → Performs the localization task
* DWB → Operates under the Controller Server
* Planner Server (optional) generates global path
* Recovery Server → Activates if cmd_vel cannot be generated

Robot Sensors and ROS2 System Architecture

* Physical Layer:
  → IMU, Odometry, LiDAR
  (Onboard robot hardware sensors)

⬇️ Sensor data is transmitted to the ROS 2 system

* Software Layer (ROS 2):
  → ROS 2 provides all communication and infrastructure

  ├── Nav2 (Navigation System)

```
    ├── Planner Server (optional, not used in this thesis)
    ├── Controller Server → Runs DWB planner
    └── Recovery Server → Executes recovery behavior if an obstacle is detected
└── AMCL → Localization (position estimation)
```

The navigation packages of ROS, Nav1 and Nav2, enable mobile robots to find the safest and most efficient path to the target. While Nav1 is compatible with ROS1, Nav2 operates under the ROS2 framework and offers a much more flexible structure.

Sensors such as IMU, odometry, and LiDAR generate raw data regarding the robot's movement, orientation, and surroundings. This data is transmitted to the software framework known as ROS 2. ROS 2 manages all communication, data flow, and inter-module connections within the system. The Nav2 navigation system also operates within this layer. The robot detects obstacles using LiDAR and performs real-time safe path planning with the DWB planner based on the current costmap. Although this structure does not directly track moving obstacles, it enables the robot to respond safely by slowing down or changing direction when obstacles are detected.

DWB and TEB Planners

In the Nav2 system, two local planners are provided by default:

- **DWB (Dynamic Window Based):**
  The ROS2 version of DWA. It selects a safe path in real time by taking into account the robot's velocity, direction, and nearby obstacles.

- **TEB (Timed Elastic Band):**
  Generates more flexible and optimal paths by also considering time. However, it has a higher computational cost.

These planners are designed to ensure that the robot continues to function smoothly even when encountering moving obstacles. In this study, only the DWB planner was used.

3.2.2.1 Environmental Representation and Costmap2D

In the ROS2 system, the environment is represented by a special structure called a costmap, which enables the robot to understand its surroundings and perform motion planning. This map numerically expresses the positions of obstacles and whether the robot can pass through specific areas. Using this map, the robot's planner and controller modules can generate a safe path around obstacles. The calculations are performed by minimizing a cost function that ensures the robot maintains a safe distance from obstacles. The costmap adds a safety margin around obstacles by assigning high costs to nearby cells to prevent direct collisions.
The costmap is a two-dimensional grid-based map of the environment. In this map:

- Each cell has a value between 0 and 255.
- This value indicates how "dangerous" the cell is for the robot.
- Internally, these values are grouped into three main categories:
  - **0 → FREE** (Free, safe to pass)
  - **254 → OCCUPIED** (Occupied by an obstacle, impassable)
  - **255 → UNKNOWN** (Unknown, no sensor data available)

The costmap defines danger zones at different levels.

JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

Table 3.4: Meaning of Cost Values

Cost Type	Meaning
Lethal	There is a direct obstacle in the cell. Impassable.
Inscribed	Very close to the robot's center. High risk of collision.
Possibly Circumscribed	Near the robot's outer frame. Collision possible depending on the turning direction.
Free Space	No obstacle in the cell. Safe to pass.
Unknown	No sensor data about the cell. Passage might be risky.
Intermediate Values	The value ranges between 0–254 depending on proximity to obstacles. Cost decreases with distance.

The costmap structure in ROS2 enables the robot to perceive the environment more intelligently and flexibly. Thanks to its layered architecture, the robot can recognize both static obstacles and moving objects more accurately and quickly. This structure provides a significant advantage for robots navigating in environments that are dynamic and populated by humans.

3.2.3 TurtleBot3

In this study, the TurtleBot3 Burger mobile robot platform was used for both simulation and real-world testing. TurtleBot3 is a small-sized, portable, and modular ROS-based robot developed specifically for academic research, education, and prototyping purposes.

The hardware of the robot consists of key components such as:

- A wheeled chassis (mobile base),
- A 360° LiDAR sensor for environmental perception,
- An IMU (accelerometer, gyroscope, magnetometer),
- Encoders (for wheel measurement).

TurtleBot3 was chosen for this study due to its compatibility with ROS 2, support for planners like Nav2 and DWB, availability of both simulation and physical testing capabilities, widespread use in academic studies, comprehensive documentation, and open-source structure.

Technical Specifications (TurtleBot3 Burger):

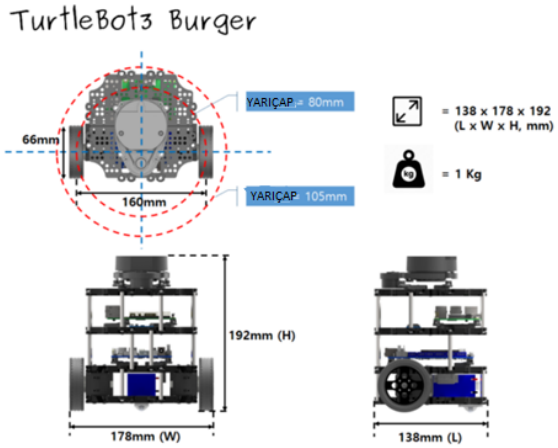


Figure 3.3 TurtleBot3 Burger Illustration

Table 3.5. Technical Specifications of TurtleBot3 Burger

Specification	Value
Maximum Speed	0.22 m/s
Payload Capacity	15 kg
Dimensions (L x W x H)	138 x 178 x 192 mm
Weight (with all components)	1 kg
Processor (MCU)	32-bit ARM Cortex-M7 (216 MHz)
SBC Used	Raspberry Pi
LiDAR Sensor	RPLIDAR A3 (used instead of LDS-01 in this thesis)

The physical structure of the robot directly affects the placement of its sensors and its perception capabilities. The TurtleBot3 platform offers a compact system equipped with LiDAR and IMU (ROBOTIS, 2017). This hardware is designed to work in integration with ROS2-based systems and serves as a suitable platform for dynamic testing (Kohlbrecher et al., 2011).

The RPLiDAR A1 sensor mounted on the robot is used to collect environmental data. This sensor performs 360° scanning and transmits the LiDAR data to the ROS 2 system, enabling the costmap to be updated.

3.3 Dynamic Path Planning

Today, with advancements in artificial intelligence for image processing and improvements in camera technologies, many robotic systems detect moving obstacles based on camera data. However, the main disadvantage of these methods is that they require high processing power, which in turn demands more expensive hardware.

Therefore, LiDAR-based systems offer a more cost-effective alternative to vision-based approaches. The required processing steps for this are:

- Filtering the LiDAR data,
- Interpreting this data as obstacles, and
- Tracking moving obstacles.

Günümüzde görüntü işleme alanındaki yapay zeka gelişmeleri ve kamera teknolojilerindeki ilerlemeler sayesinde, birçok robot sistemi **hareketli engelleri kamera verilerine dayanarak algılamaktadır**. Ancak bu yöntemlerin en büyük dezavantajı, yüksek işlem gücü gerektirmeleri ve dolayısıyla daha **pahalı donanımlar** gerektirmeleridir.

Bu nedenle daha **ekonomik** bir çözüm olan **LiDAR tabanlı sistemler**, görüntü işlemeye alternatif olabilir. Bunun için gereken işlem adımları:

- LiDAR verilerinin filtrelenmesi,
- Bu verilerin **engel olarak yorumlanması** ve
- Hareketli engellerin **takip edilmesi (tracking)** sürecidir.

3.3.1 LiDAR-Based Planning Solutions

Moving obstacles are mostly handled at the local planning level. In ROS2 Nav2, this is done through the Controller Server.

DWB Controller (Dynamic Window Based):

- Evaluates the combinations of translational and rotational velocities of the robot.
- Calculates all possible velocities the robot can achieve starting from its current velocity.
- Eliminates velocity combinations that would lead to a collision.
- Selects the most suitable one among the remaining options using a cost function.
- This cost considers factors such as the robot’s orientation toward the goal, distance from obstacles, and current speed.

TEB Controller (Timed Elastic Band):

- The path is divided into small segments, and each segment behaves like an elastic band that deforms to avoid obstacles.

- In this system, time information is also assigned to each path segment. Thus, both time and distance are optimized.

Real Application with TEB

Figure 3.4: Testing of the TEB Method in a University Environment

The usability of TEB was tested in an experimental study. The TEB-based test scene used in this study was directly taken from the structure of the Marathon 2 system by Macenski et al. (2020). In a university corridor filled with people, two robots were guided using Nav2 and TEB. Although the robots reached their targets, they exhibited recovery behavior with an average duration of 4.3 seconds.

The causes of this situation were poor localization due to LiDAR beams being blocked by people, and the need for re-planning because moving individuals crossed the robot's path. In such cases, the robot generally switched to a "wait" command, waiting for the obstacle to move away before continuing.

Conclusion:

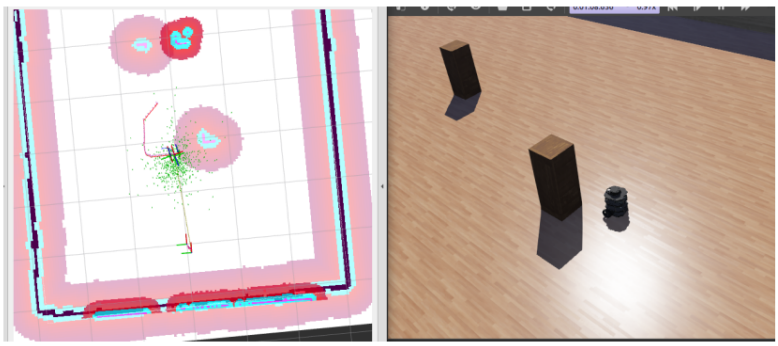


Figure 3.5 Map Representation of Obstacles in the Simulation Environment

Although TEB indirectly accounts for dynamic obstacles, it does not employ a specialized structure that explicitly detects them. For mobile robots to reach their goals in dynamic environments without collisions, real-time detection and interpretation of surrounding obstacles is critical (Fox et al., 1997). In this context, the Dynamic Window Approach allows the robot to determine collision-free paths in accordance with its instantaneous speed and turning angles (Seder & Petrović, 2007).

JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

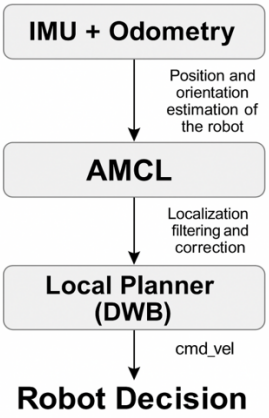
Table 3.7 Comparison of Dynamic Path Planning Methods

Feature / Algorithm	DWB	TEB (Timed Elastic Band)
Obstacle Detection	Detects static obstacles	Perceives as static obstacles (no specific support for dynamic ones)
Dynamic Obstacle Sensitivity	None	Indirect, no specific reaction
Path Planning	Step-by-step decisions toward the goal	Plans the entire path flexibly from start to end
Waiting Behavior	Can stop if there is an obstacle	May wait if the obstacle is too close and path is blocked
Navigation Duration	Moderate	Longest (many recovery behaviors)
Path Naturalness	Moderate (may include short turns)	High (smooth and natural turns)
Computational Load	Low	High (includes optimization solving)
Performance in Complex Environments	Moderate	Moderate to high (higher recovery rate)

3.3.2 Methods

The IMU and odometry data collected from the robot's own sensors serve as the primary inputs for estimating the robot's position. However, the exact position is calculated by the AMCL algorithm within the ROS 2 system. In other words, raw hardware data is refined and processed by software. The method consists of three main steps:

1. Object Detection
2. Cost Assignment



In this study, the method used to enable autonomous movement of the robot consists of three fundamental steps where both hardware and software components operate in coordination. First, data obtained from the robot's IMU and odometry sensors is used to estimate the robot's current position and orientation. This estimation is refined by the AMCL algorithm running in the ROS 2 system. AMCL interprets the sensor data on the map and provides a corrected estimate of the robot's true location.

This positional information is then transferred to the local planner algorithm DWB (Dynamic Window Approach). DWB analyzes the robot's current velocity, heading, and obstacle data from the costmap to compute the safest motion command (cmd_vel). This command is then sent directly to the robot, allowing it to move toward the goal while avoiding collisions.

Only the DWB algorithm was utilized in this thesis. The system operates based solely on current sensor data, taking into account both static and immediate environmental changes.

3.3.2.1 Object Detection

In this study, the detection of moving objects was directly carried out through the costmap (cost map) representation. The robot's surroundings are perceived via the LiDAR sensor, and this data is projected onto the costmap. Each costmap cell is monitored over time, and the change between two consecutive updates is

evaluated. If the change exceeds a certain threshold, the cell is considered to represent a moving object. This method can be represented by the following formula:

F(t) = |C(t) − C(t−1)| > c
Motion_Difference = |Current_Value − Previous_Value| > Threshold

Here, F(t) is the condition for determining motion; C(t) is the current costmap cell value, C(t−1) is the previous value, and *c* is a predefined threshold in the system. Cells flagged with this logic are grouped together and interpreted as moving objects. In this study, the position tracking or speed estimation of the moving objects was not performed.

3.3.2.2 Cost Assignment

After detecting moving objects, the potential hazards they may pose to the robot are represented on the costmap. For this purpose, a two-dimensional Gaussian function is applied around each moving obstacle, considering its direction and speed. Thus, the surroundings of the obstacle are extended with a risk zone that decreases based on distance. Fast-moving objects are assigned larger risk areas; especially in the direction of movement, the spread is greater, while it is less behind the object. This structure enables the robot to be more sensitive to potential threats in front of it. This approach is inspired by human behavior-based social space modeling. Mathematically, the risk (cost) value of a cell is calculated to decrease according to its distance from the obstacle. The Gaussian-based formula used is as follows:

Formula:
$C(q) = A \cdot \exp(-d^2 / 2\sigma^2)$

$\text{Cost} = \text{Max value} \times \exp(-\text{distance}^2 / 2\text{spread}^2)$

Here, C(q) represents the cost value of the cell; d is the distance between the cell and the obstacle; σ represents the spread coefficient. As a result, the system analyzes environmental changes using only LiDAR data and transfers this information to the DWB local planner to ensure real-time and safe path planning.

3.4 APPLICATION

In this section, how the proposed approach is implemented under the ROS2 framework within the scope of this thesis is explained in general terms. In this section, how the developed system was implemented under the ROS 2 platform is explained. During the application process, the Linux-based Ubuntu 20.04 operating system was used. Due to performance problems encountered in virtual machines, the system was installed directly on a physical computer. The Humble distribution of ROS 2 was installed using Debian packages from official sources.
During the development process, two main workspaces were created:

- **turtlebot3_ws**: Contains the basic driver and simulation packages for the TurtleBot3 robot. The robot model was used both in simulation and on real hardware via this workspace.

- **ros2_ws**: The main workspace containing the custom software and ROS 2 extensions developed within the scope of the thesis.

Applications were tested in the Gazebo simulation environment, and RViz interface was used for visualizing the outputs.

3.4.1 Messages and Topics

Custom ROS message structures were defined for the detection and tracking of dynamic obstacles:

- **Obstacle.msg**: Contains identification (UUID), position, velocity, and size information for each obstacle.

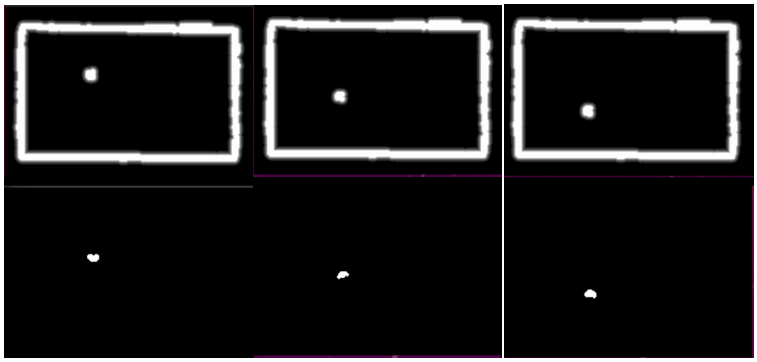
- **ObstacleArray.msg**: Carries the list of all simultaneously detected obstacles.

These messages were transferred to other components of the system via the */detection* and */local_costmap/tracking* topics, respectively. Velocity information was added only during the tracking phase.

3.4.2 Costmap Transformation (Obstacle Detection)

Dynamic obstacle detection is performed directly based on the costmap data generated by the robot. This map, created using information obtained from the LiDAR sensor, analyzes time-varying cells to identify moving objects. Each moving cell is evaluated in groups and defined as a single object. The detected objects are then published in the **Obstacle.msg** format. With this approach, inspired by methods developed on graph-based optimization and data representation, dynamic obstacles are extracted from costmap data (Kümmerle et al., 2011).

The figure below illustrates how an obstacle is detected over time:



(a) $t = 1s$. (b) $t = 2s$ (c) $t = 3s$

Figure 3.6 Detection of obstacles over time

3.4.3 Obstacle Detection and Costmap Update

In this study, the detection of obstacles in the environment and the modeling of environmental risks were carried out directly on the costmap data. The environment of the robot is scanned using a LiDAR sensor, and the moving obstacles are reflected on the costmap using this sensor data.

Time-varying cell values are analyzed based on a predefined threshold, and obstacle regions are identified. These identified areas are evaluated as potential danger zones around the robot.

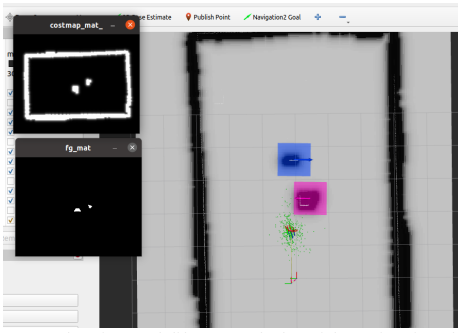


Figure 3.7 Working mechanism of filters and algorithms in simulation environment

3.4.4 Dynamic Costmap Layer

Based on the detected moving obstacle regions, risk zones were created around the robot. This process is intended to enhance the robot's environmental awareness and support safe path planning.
Each obstacle is surrounded by a Gaussian-based 2D function centered on its position, expanding the area and assigning higher cost values.
This allows the robot to generate routes that are more sensitive and safer, especially in response to obstacles that suddenly appear in front of it.

JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

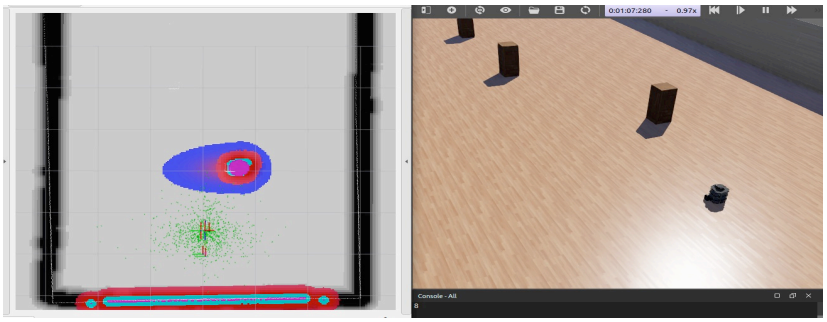


Figure 3.8 Visualization of dynamic costmap layers in simulated environment

4. RESEARCH RESULTS AND DISCUSSION

4.1 Simulation Environment and Tests

ROS2 is an open-source platform that enables modular development of robot software (Quigley et al., 2009). Within this architecture, the Nav2 package works with local and global planners to perform path planning in dynamic environments (Macenski et al., 2020). The physical simulation environment of the robot is supported by Gazebo software, allowing real-time testing (Koenig & Howard, 2004).

The simulation studies were carried out in the Gazebo environment using the ROS2 Nav2 package and the TurtleBot3 Burger robot model. The aim was to analyze the impact of static and dynamic obstacles on path planning and observe how the costmap configuration guides this planning. Tests were conducted under three different scenarios:

1. In the first scenario, the robot was directed to fixed target points on a map without obstacles. This test was performed to establish baseline time and distance reference values.

2. In the second scenario, physical obstacles (tables, chairs, etc.) were added to the map, and the robot was directed to the same targets. The robot's route changes and deviation amounts in response to these newly detected obstacles were analyzed.

3. In the third scenario, new obstacles were both simulated and costmap parameters were altered. By narrowing the costmap width, the robot was allowed to pass closer to obstacles. This configuration resulted in a more flexible but riskier path.

4.1.1 Dynamic Obstacle Scenario

In this study, the robot was first directed toward fixed targets on a pre-known map created using SLAM. Then, new obstacles were manually added to the map to observe the system's response. These obstacles were not initially present on the map but were introduced into the scene during simulation.

The robot detected these new obstacles using the LiDAR sensor and replanned its route via the costmap. This demonstrated the system's sensitivity to environmental changes regardless of whether the obstacle was moving.

Since the timing and position of the added obstacles were manually determined by the user, the scenario required an instant reaction from the robot—similar to encountering a moving object. In this regard, the system's dynamic obstacle avoidance capability was evaluated.

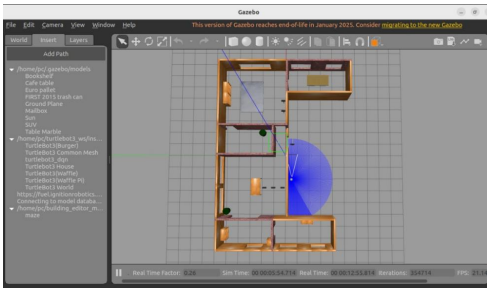


Figure 4.1 Gazebo Simulation

JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

4.1.2 RViz Observations

In the observations made via the RViz interface, the robot's costmap updates were monitored in real-time. During the simulations, deviations along the robot's path, arrival times to the target, and obstacle avoidance maneuvers were recorded in detail. Especially in the third test scenario, it was observed that the robot reached the target more quickly after costmap parameter changes.

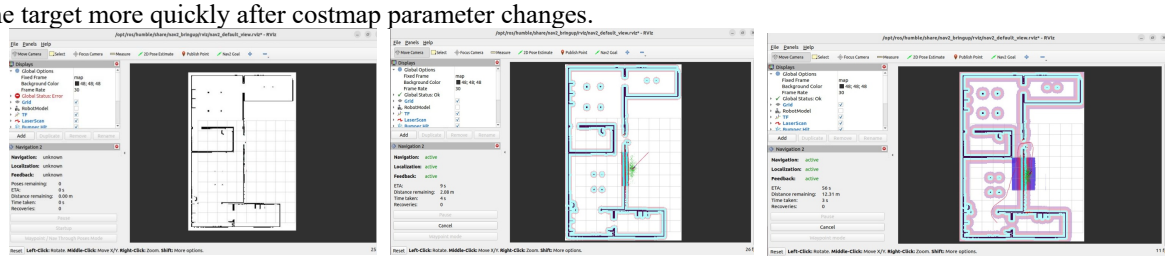


Figure 4.2 RViz Maps

4.2 SLAM (Simultaneous Localization and Mapping)

In order for the robot to estimate its position on a given map, the AMCL (Adaptive Monte Carlo Localization) algorithm, which is the default in the ROS2 Nav2 system, was utilized. AMCL uses a particle filter to estimate the robot's location on a known map. In the tests, the robot was localized using this method before being directed to the target.

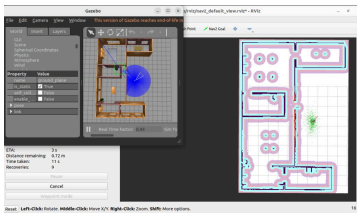
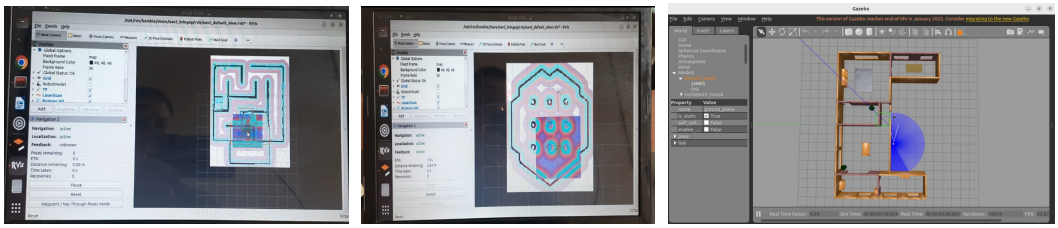


Figure 4.3 Simultaneous Simulation Image

4.3 Simulation Tests

The simulation tests were conducted on a fixed map using three different configurations. Each square has an edge length of 1 meter, and the costmap ensures a minimum radius of 20 cm around each obstacle in order to prevent the robot from colliding. In these configurations, both the obstacle types and the costmap settings were modified. In each setup, the robot was directed to the same target points, and the following variables were compared: path deviation, time to reach the goal, and total distance traveled. The results are summarized in the table below.



JOURNAL OF NATURAL SCIENCES AND TECHNOLOGIES
2025, 4(1), pp. 345-364, DOI: 10.5281/zenodo.16410654

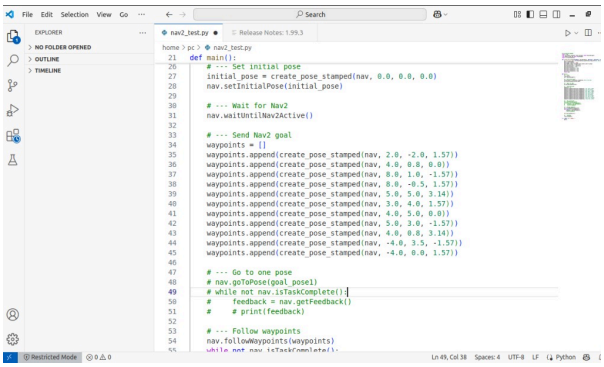


Figure 4.4 Python Software Coordinates

Table 4.1 Performance Metrics of Route Planning Methods

Environment Type	Obstacle Type	Costmap Update	Route Changed?	Path Length	Duration
Office	Static	No	No	38 m	20 min
Office	Moving Target	No	Yes	42 m	24 min
Office	Moving Target	Yes	Yes	36 m	19 min
Maze	Static	No	No	15	105 sec
Maze	Moving Target	No	No	15	107 sec
Maze	Moving Target	Yes	No	15	103 sec
Hexagonal	Static	No	No	10	120 sec
Hexagonal	Moving Target	No	Yes	16	140 sec
Hexagonal	Moving Target	Yes	Yes	10	110 sec

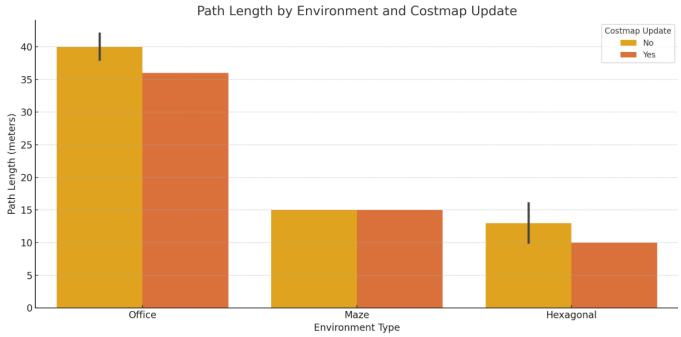


Figure 4.5. The Impact of Costmap Updates on Path Length in Different Environments

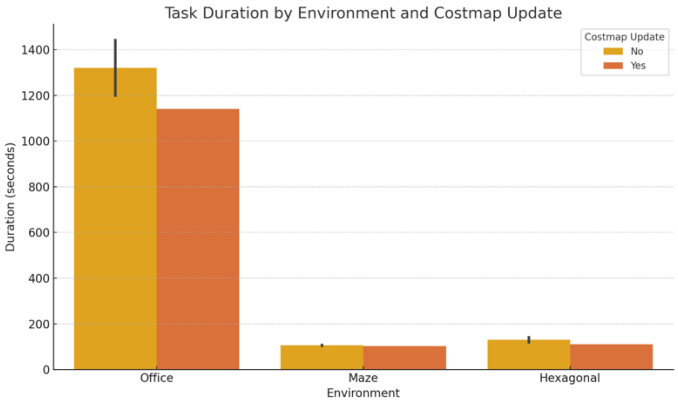


Figure 4.6. The Impact of Costmap Updates on Task Duration in Different Environments

4.4 Discussion

According to the test results, the robot's tendency to change its route increases in environments with moving obstacles. While the route generally remains stable in scenarios with static obstacles, the real-time update of moving obstacles to the costmap forces the robot to replan its path. Notably, by modifying the costmap parameters, the robot was able to select shorter paths and reach the target more quickly. This demonstrates the significant impact of costmap configuration on overall system performance. Variables such as environment type, complexity level, and frequency of obstacle movement directly influence both route and time performance.

5. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, a ROS2-based path planning approach was implemented to enable mobile robots to reach their targets safely and flexibly in indoor environments with dynamic obstacles. The developed system utilized the default local planner of the Nav2 framework, the DWB (Dynamic Window Based) controller. The main objective was to allow the robot to exhibit basic adaptive responses to real-time environmental changes.

Simulation tests were conducted in the Gazebo environment using the TurtleBot3 robot. During these tests, an obstacle that was not initially present on the map was added along the robot's route to the goal, and the robot's response was observed. It was noted that objects of a certain height could be detected by the LiDAR sensor, enabling the creation of an updated environmental model on the costmap. Based on this model, the DWB controller was able to generate a new path.

Once the obstacle was detected by the LiDAR, the robot's costmap data was dynamically updated, and the DWB planner performed a new path planning process using this information. This allowed the robot to change its route before any collision occurred, resulting in a successful navigation sequence. Through this architecture, the system gained fundamental flexibility to respond to environmental changes without solely relying on a static map.

Simulation results demonstrated that the DWB controller could replan based on sensor input in response to environmental changes and reach the goal successfully. The quick engagement of the robot's decision-making mechanism prevented collisions and enhanced navigation performance.

During the navigation process, the LiDAR sensor was used for environmental awareness, while the robot's position and movement data were obtained from IMU sensors and evaluated within the system. These data were integrated using the AMCL algorithm in the ROS2 framework, forming a robust localization structure.

As a result, the DWB-based approach implemented in this study provided a fast and reliable response to basic environmental changes, combining environmental perception with path planning to achieve a balanced and collision-free navigation process. This system offers an effective alternative, particularly in environments with sudden environmental changes, by enabling rapid route updates with minimal delay.

6. CONFLICT OF INTEREST

The authors declare that there are no conflicts of interest regarding the publication of this paper.

7. FUNDING / ACKNOWLEDGEMENTS

No funding or research grants were received during the preparation of this study. This paper includes the experimental results of the MSc thesis study of Cem Berkay KARACA.

8. DATA AVAILABILITY

The following are the video links demonstrating the experimental results of this study:
Office Environment 1

Office Environment 2

Office Environment 3

Maze Environment

Hexagonal Environment

REFERENCES

1. Bar-Shalom, Y., Li, X. R., &Kirubarajan, T. (2001). Estimationwith Applications toTrackingandNavigation. Wiley.

2. Borenstein, J., &Koren, Y. (1991). Thevectorfield histogram—Fastobstacleavoidancefor mobile robots. IEEE Transactions on RoboticsandAutomation, 7(3), 278–288.
3. Grisetti, G., Kummerle, R., Stachniss, C., &Burgard, W. (2010). A tutorial on graph-based SLAM. IEEE IntelligentTransportationSystems Magazine, 2(4), 31–43.
4. Julier, S. J., &Uhlmann, J. K. (2004). Unscentedfilteringandnonlinearestimation. Proceedings of the IEEE, 92(3), 401–422.
5. Koenig, N., & Howard, A. (2004). Design anduseparadigmsforGazebo, an open-sourcemulti-robot simulator. InProceedings of the 2004 IEEE/RSJ International Conference on IntelligentRobotsandSystems (Vol. 3, pp. 2149–2154). IEEE.
6. Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingner, M., Bormann, R., ... &vonStryk, O. (2011). A flexibleandscalable SLAM systemwithfull 3D motionestimation. In 2011 IEEE Safety, Security, andRescueRobotics (SSRR) (pp. 155–160). IEEE.
7. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., &Burgard, W. (2011). g2o: A general frameworkforgraphoptimization. In 2011 IEEE International Conference on RoboticsandAutomation (pp. 3607–3613). IEEE.
8. Macenski, S., Martín, F., White, R., &Ginés, R. (2020). TheMarathon 2: A NavigationSystem. IEEE/RSJ International Conference on Intelligent Robotsand Systems (IROS).
9. Moravec, H., &Elfes, A. (1985). High resolution maps from wideangle sonar. InProceedings. 1985 IEEE International Conference on Roboticsand Automation (Vol. 2, pp. 116–121). IEEE.
10. O'Kane, J. M. (2013). A GentleIntroductionto ROS.
11. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., & Ng, A. Y. (2009). ROS: An open-source Robot Operating System. ICRA Workshop on Open Source Software, 3(3.2), 5.
12. ROBOTIS. (2017). TurtleBot3 Technical Specifications.
13.Seder, M., &Petrović, I. (2007). Dynamic window based approach to mobile robot motioncontrol in the presence of movingobstacles. *Proceedings of the 2007 IEEE International Conference on RoboticsandAutomation (ICRA)*, 1986–1991. IEEE.
14. Zhang, J., & Singh, S. (2014). LOAM: LidarOdometryandMapping in Real-time. In Robotics: Scienceand Systems.